CSE443 – Object Oriented Design and Analysis Course

# Report

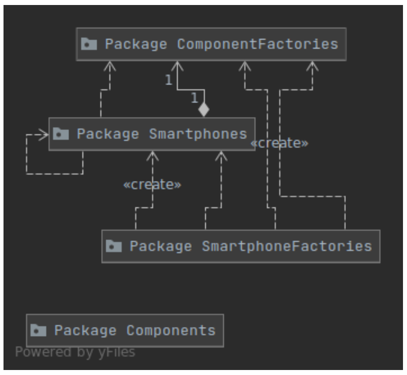
Midterm Project

OĞUZHAN AGKUŞ
161044003

## Preface

I used Intellij Idea IDE for implemantation. I assume that course is main project and each homework/sub-project is a module of this project. Also each sub-project is a Java package. So, the IDE have its own representations. I will not upload whole project file, I will upload only source code. There is a slight possiblity to my source codes will give some errors. I am not sure about that. If it gives some errors, its reason is this. My codes compiles without errors and run successully. I will add output screenhots for each part. Each part has own demo code, own diagrams and javadoc at its directory.

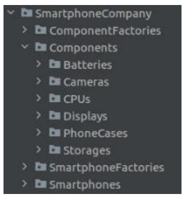
## Question 1 – Smartphone Factory



I used the Abstract Factory design pattern. I decided to split into 4 group all my classes. They are Smartphones, SmartphoneFactories, Components and ComponentFactories. Because there are a lot of classes, I just wanted to organize them. To do this, I create sub packages.

These packages have these relations: ComponentFactories produces Components, Smartphones consist of Components, SmartphoneFactories produces Smartphones by combining Components.

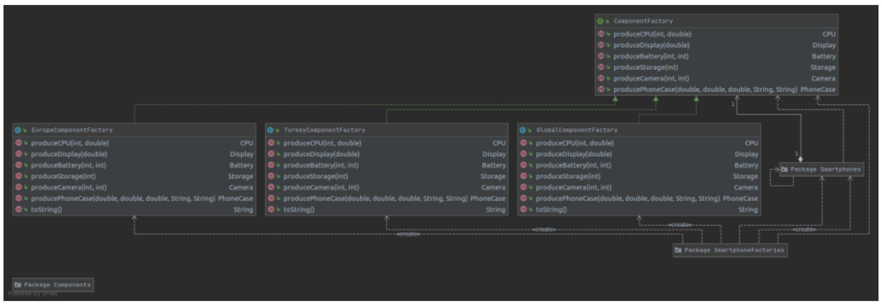
Components are CPU, RAM, Display, Battery, Storage, Camera and Case. They are defined as interface because there are more then one types of these components. They implemets these abstract class. Each abstract class and their concrete classes are combined into a package. My package hierarchy is like:



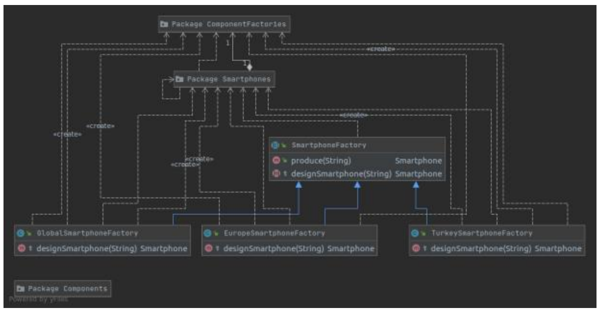
```
SmartphoneCompany
  ComponentFactories
  Components
    Batteries
    Cameras
    CPUs
    Displays
    PhoneCases
    Storages
  SmartphoneFactories
  Smartphones
```

The concrete classes of components are implemented according to market requirements. For example, 8-cores, 4-cores, and 2-cores CPUs are classes that implement CPU abstract class. Another example, LithiumBoron, LithiumIon, and LithiumCobalt batteries are classes that implements Battery abstract class.

ComponentFactory interface implemented in 3 different way. Each implementation is for a market. They produces components according to market requirements.

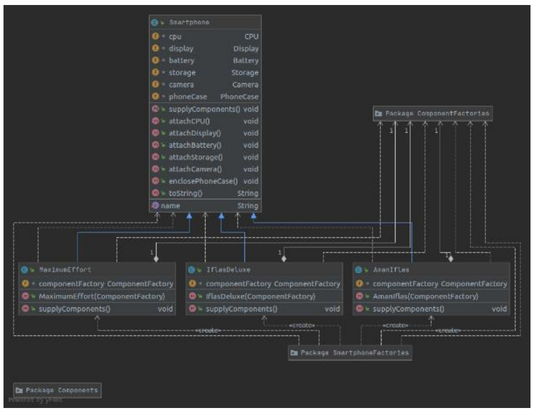


ComponentFactory
produceCPU(int, double) CPU
produceDisplay(double) Display
produceBattery(int, int) Battery
produceStorage(int) Storage
produceCamera(int, int) Camera
producePhoneCase(double, double, double, String, String) PhoneCase
EuropeComponentFactory
TurkeyComponentFactory
GlobalComponentFactory
toString() String
Package Smartphones
«create»
Package SmartphoneFactories
Package Components

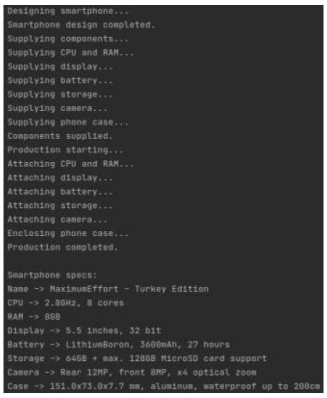


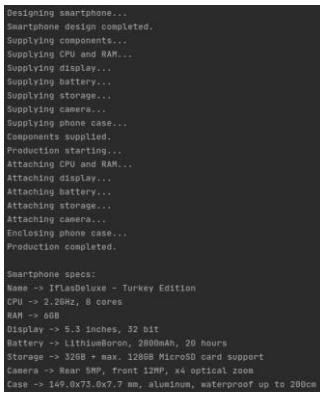
SmartphoneFactory is a abstract class. It only implements produce method which incidates the production phase. It should be implemented in this class because this phase will not change. Only design phase will change according to model and target. So the function does that is abstract. Concrete classes must define design phase. There are 3 different implementation of this class.

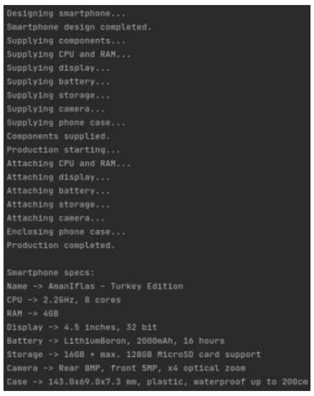
Smartphone is defined as abstract class. Because Smartphones has certain properties which will not change model to model. Each model derived from this class. And they must implement supplyComponents method which prepare the components according to model and market.

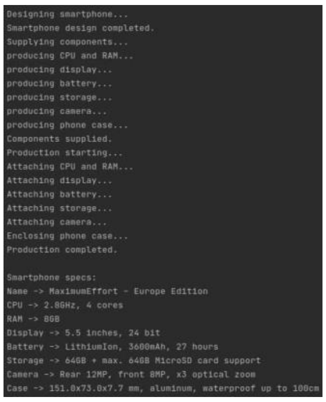


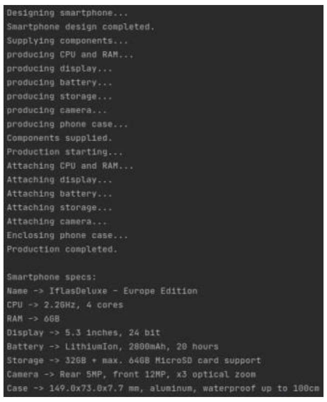
The demo code produces 3x3=9 different smartphone.

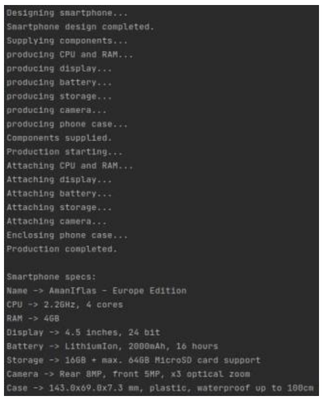

```
Designing smartphone...
Smartphone design completed.
Supplying components...
Supplying CPU and RAM...
Supplying display...
Supplying battery...
Supplying storage...
Supplying camera...
Supplying phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> MaximumEffort - Turkey Edition
CPU -> 2.8GHz, 8 cores
RAM -> 8GB
Display -> 5.5 inches, 32 bit
Battery -> LithiumBoron, 3600mAh, 27 hours
Storage -> 64GB + max. 128GB MicroSD card support
Camera -> Rear 12MP, front 8MP, x4 optical zoom
Case -> 151.0x73.0x7.7 mm, aluminum, waterproof up to 200cm
```

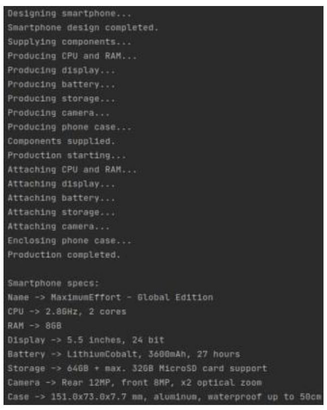

```
Designing smartphone...
Smartphone design completed.
Supplying components...
Supplying CPU and RAM...
Supplying display...
Supplying battery...
Supplying storage...
Supplying camera...
Supplying phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> IflasDeluxe - Turkey Edition
CPU -> 2.2GHz, 8 cores
RAM -> 6GB
Display -> 5.3 inches, 32 bit
Battery -> LithiumBoron, 2800mAh, 20 hours
Storage -> 32GB + max. 128GB MicroSD card support
Camera -> Rear 5MP, front 12MP, x4 optical zoom
Case -> 149.0x73.0x7.7 mm, aluminum, waterproof up to 200cm
```

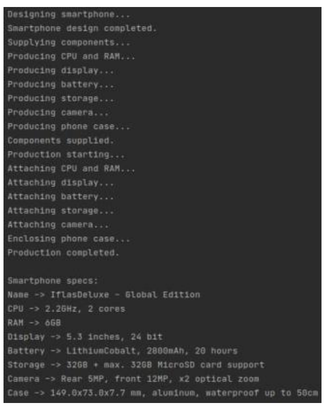

```
Designing smartphone...
Smartphone design completed.
Supplying components...
Supplying CPU and RAM...
Supplying display...
Supplying battery...
Supplying storage...
Supplying camera...
Supplying phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> AmanIflas - Turkey Edition
CPU -> 2.2GHz, 8 cores
RAM -> 4GB
Display -> 4.5 inches, 32 bit
Battery -> LithiumBoron, 2000mAh, 16 hours
Storage -> 16GB + max. 128GB MicroSD card support
Camera -> Rear 8MP, front 5MP, x4 optical zoom
Case -> 143.0x69.0x7.3 mm, plastic, waterproof up to 200cm
```

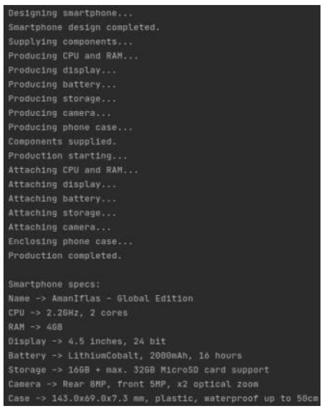

```
Designing smartphone...
Smartphone design completed.
Supplying components...
producing CPU and RAM...
producing display...
producing battery...
producing storage...
producing camera...
producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> MaximumEffort - Europe Edition
CPU -> 2.8GHz, 4 cores
RAM -> 8GB
Display -> 5.5 inches, 24 bit
Battery -> LithiumIon, 3600mAh, 27 hours
Storage -> 64GB + max. 64GB MicroSD card support
Camera -> Rear 12MP, front 8MP, x3 optical zoom
Case -> 151.0x73.0x7.7 mm, aluminum, waterproof up to 100cm
```


```
Designing smartphone...
Smartphone design completed.
Supplying components...
producing CPU and RAM...
producing display...
producing battery...
producing storage...
producing camera...
producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> IflasDeluxe - Europe Edition
CPU -> 2.2GHz, 4 cores
RAM -> 6GB
Display -> 5.3 inches, 24 bit
Battery -> LithiumIon, 2800mAh, 20 hours
Storage -> 32GB + max. 64GB MicroSD card support
Camera -> Rear 5MP, front 12MP, x3 optical zoom
Case -> 149.0x73.0x7.7 mm, aluminum, waterproof up to 100cm
```


```
Designing smartphone...
Smartphone design completed.
Supplying components...
producing CPU and RAM...
producing display...
producing battery...
producing storage...
producing camera...
producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> AmanIflas - Europe Edition
CPU -> 2.2GHz, 4 cores
RAM -> 4GB
Display -> 4.5 inches, 24 bit
Battery -> LithiumIon, 2000mAh, 16 hours
Storage -> 16GB + max. 64GB MicroSD card support
Camera -> Rear 8MP, front 5MP, x3 optical zoom
Case -> 143.0x69.0x7.3 mm, plastic, waterproof up to 100cm
```

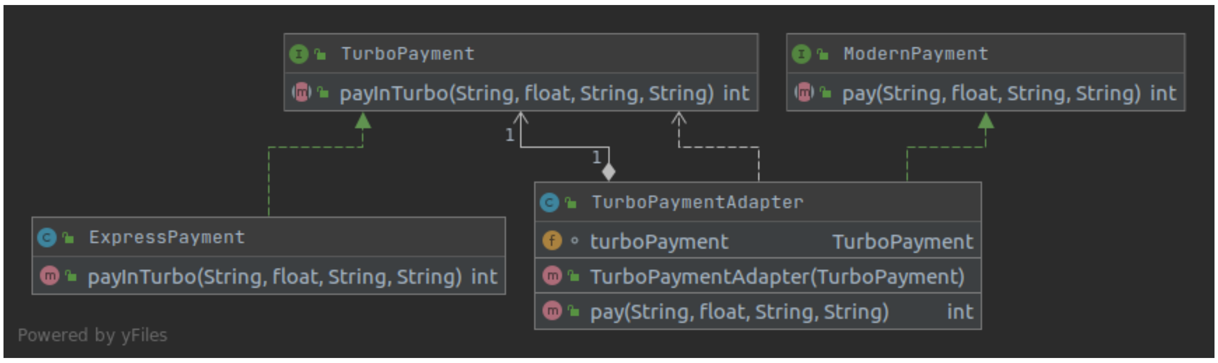

```
Designing smartphone...
Smartphone design completed.
Supplying components...
Producing CPU and RAM...
Producing display...
Producing battery...
Producing storage...
Producing camera...
Producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> MaximumEffort - Global Edition
CPU -> 2.8GHz, 2 cores
RAM -> 8GB
Display -> 5.5 inches, 24 bit
Battery -> LithiumCobalt, 3600mAh, 27 hours
Storage -> 64GB + max. 32GB MicroSD card support
Camera -> Rear 12MP, front 8MP, x2 optical zoom
Case -> 151.0x73.0x7.7 mm, aluminum, waterproof up to 50cm
```


```
Designing smartphone...
Smartphone design completed.
Supplying components...
Producing CPU and RAM...
Producing display...
Producing battery...
Producing storage...
Producing camera...
Producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> IflasDeluxe - Global Edition
CPU -> 2.2GHz, 2 cores
RAM -> 6GB
Display -> 5.3 inches, 24 bit
Battery -> LithiumCobalt, 2800mAh, 20 hours
Storage -> 32GB + max. 32GB MicroSD card support
Camera -> Rear 5MP, front 12MP, x2 optical zoom
Case -> 149.0x73.0x7.7 mm, aluminum, waterproof up to 50cm
```


```
Designing smartphone...
Smartphone design completed.
Supplying components...
Producing CPU and RAM...
Producing display...
Producing battery...
Producing storage...
Producing camera...
Producing phone case...
Components supplied.
Production starting...
Attaching CPU and RAM...
Attaching display...
Attaching battery...
Attaching storage...
Attaching camera...
Enclosing phone case...
Production completed.

Smartphone specs:
Name -> AmanIflas - Global Edition
CPU -> 2.2GHz, 2 cores
RAM -> 4GB
Display -> 4.5 inches, 24 bit
Battery -> LithiumCobalt, 2000mAh, 16 hours
Storage -> 16GB + max. 32GB MicroSD card support
Camera -> Rear 8MP, front 5MP, x2 optical zoom
Case -> 143.0x69.0x7.3 mm, plastic, waterproof up to 50cm
```

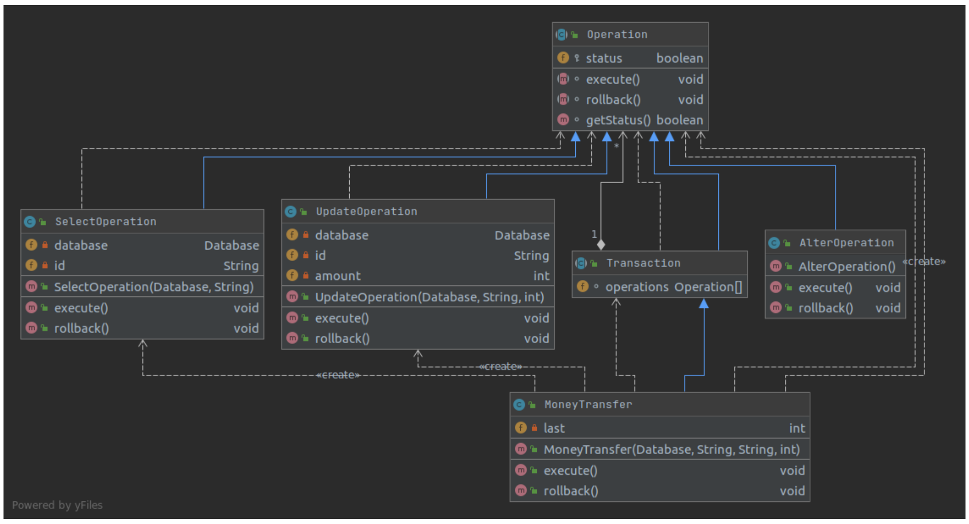
## Question 2 – Adapter

This question needs to use Adapter design pattern. We just need a class that implements ModernPayment interface. That class will be a wrapper for TurboPayment objects.


TurboPayment
payInTurbo(String, float, String, String) int
ModernPayment
pay(String, float, String, String) int
1
1
TurboPaymentAdapter
turboPayment TurboPayment
TurboPaymentAdapter(TurboPayment)
pay(String, float, String, String) int
ExpressPayment
payInTurbo(String, float, String, String) int
Powered by yFiles

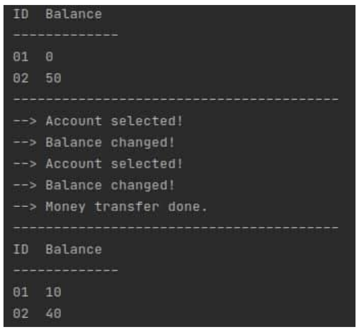
## Question 3 – Transaction

My approach to solve this problem is using Command design pattern.

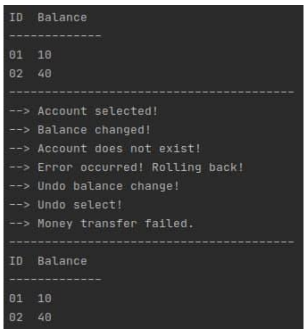


The operation must have 2 methods, execute and rollback. Also I need one more method to check if operation succeed. Each operation implements their execute and rollback methods.

A transansaction is consist of some operations. I designed MoneyTransfer transcation which contains SELECT and UPDATE operations. In execute method of the transaction, it executes each operation sequentially and checks if it is succeed. If an operation fails, it stops to execute new commands and start to call rollback methods of succeed operations. To follow last executed operation, it stores an variable. I defined simple database class to test my solution approach. There are two outputs:



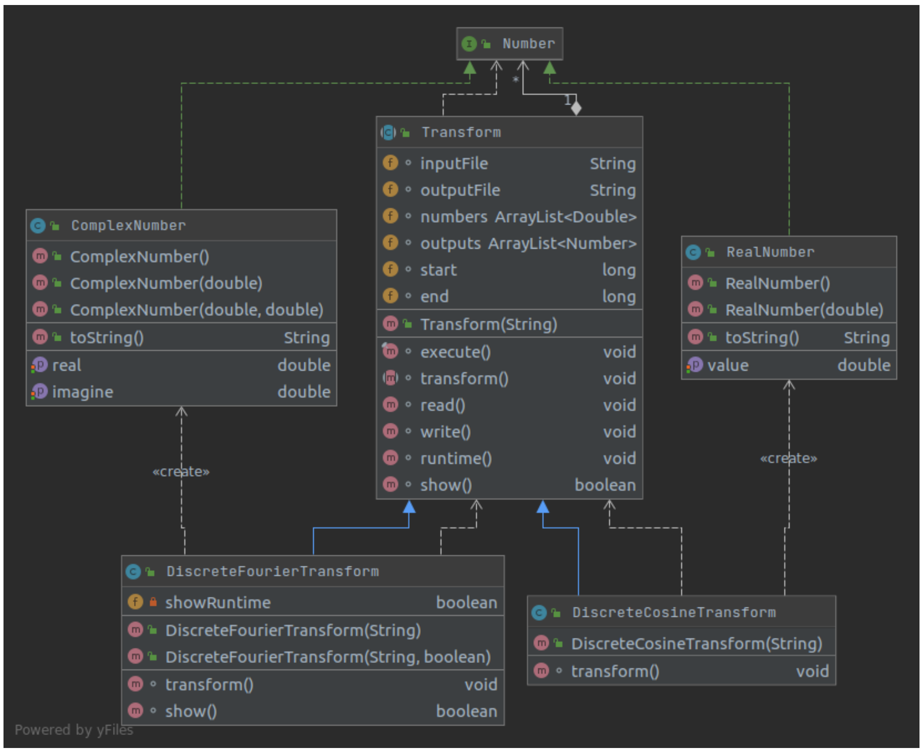
```
ID  Balance
-------------
01  0
02  50
----------------------------------------
--> Account selected!
--> Balance changed!
--> Account selected!
--> Balance changed!
--> Money transfer done.
----------------------------------------
ID  Balance
-------------
01  10
02  40
```



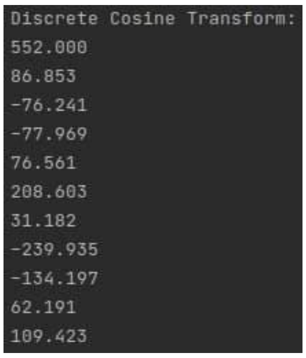
```
ID  Balance
-----------
01  10
02  40
----------------------------------------
--> Account selected!
--> Balance changed!
--> Account does not exist!
--> Error occurred! Rolling back!
--> Undo balance change!
--> Undo select!
--> Money transfer failed.
----------------------------------------
ID  Balance
-----------
01  10
02  40
```

## Question 4 – Transformer

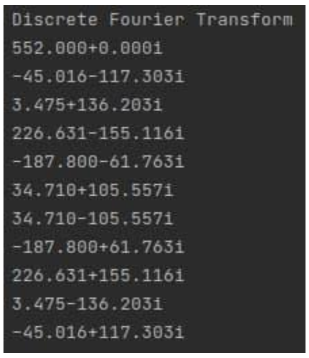
I used Template design pattern for this question. I created abstract class Transform. It defines the template. Only one step can change, this step is calculating transform. So, the method that will do this should be abstract and the concrete classes must implement it. There are another problem, about outputs of DCT and DFT. DCT produces real numbers but DFT create complex numbers. So I defined another interface called Number. RealNumber and Complex number classes implements this interface. The transform objects store the output as Number lists.



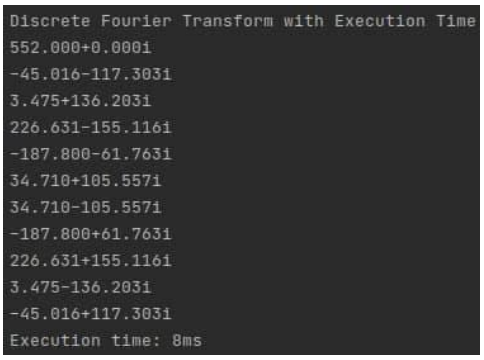
Also there is a hook method. It is not used as default but, DFT objects can activate it and print the runtime. There are my sample outputs for same inputs:



```
Discrete Cosine Transform:
552.000
86.853
-76.241
-77.969
76.561
208.603
31.182
-239.935
-134.197
62.191
109.423
```



```
Discrete Fourier Transform
552.000+0.000i
-45.016-117.303i
3.475+136.203i
226.631-155.116i
-187.800-61.763i
34.710+105.557i
34.710-105.557i
-187.800+61.763i
226.631+155.116i
3.475-136.203i
-45.016+117.303i
```



```
Discrete Fourier Transform with Execution Time
552.000+0.000i
-45.016-117.303i
3.475+136.203i
226.631-155.116i
-187.800-61.763i
34.710+105.557i
34.710-105.557i
-187.800+61.763i
226.631+155.116i
3.475-136.203i
-45.016+117.303i
Execution time: 8ms
```